

# Clase 5 — 28.11.2025

#JAVA

#androidstudio



Profesor: Joan Salvador Gordi Ortega



Programación Multimedia y Dispositivos Móviles



Clase 5— 28/11/2025



Tema: Primer Proyecto Android Studio

## 1. Inicio del proyecto — ¿Qué estamos creando realmente?

Cuando abres Android Studio y seleccionas **Empty Views Activity**, estás pidiendo que Android Studio genere por ti la estructura mínima que necesita cualquier App Android moderna:

- Una Activity principal (MainActivity.java)
- Un layout asociado (activity\_main.xml)
- Un AndroidManifest configurado
- Un proyecto Gradle funcional
- Un árbol de recursos `res/`
- El namespace único de tu aplicación

Justamente lo que aparece en el PDF de ejemplo:

Estructura de un Proyecto Android ... 1. AndroidManifest.xml ... 2. Layout XML ... 3. Activity Java ...

El profesor recalca que Android Studio añade **plantillas** para que no partamos desde cero.

## 2. Configuración inicial del proyecto — Nombre, paquete, SDK y lenguaje

En las capturas de tu clase vemos que al crear el proyecto introduces algo como:

```
com.example.digitech.dam.pmpdp.p001
```

Esto responde al estándar **reverse domain name notation**.

### 2.1 ¿Por qué este estándar?

El profesor lo explica:

*“Para evitar colisión de nombres entre aplicaciones.”*

Exacto: cuando subes apps al ecosistema Android, Google Play necesita que **cada app del planeta tenga un identificador único**.

Si tú llamas tu paquete `com.juan.app1` y otro desarrollador usa la misma combinación, habría conflicto.

El dominio invertido garantiza unicidad porque:

- El dominio pertenece a la empresa → nadie más puede usarlo
- Se añade el resto del árbol según organización interna

Ejemplo típico empresarial:

```
es.digitech.apps.calculadora
es.digitech.apps.crm
es.digitech.apps.medicion
```

---

## 2.2 Minimum SDK — ¿Qué es y por qué importa?

Al crear el proyecto la plantilla pide:

**Minimum SDK** = versión mínima de Android compatible.

En tu captura seleccionas:

| API 24 (“Nougat”, Android 7.0)

Y Android Studio te muestra:

| “Your app will run on approximately 98.6% of devices.”

Explicación ampliada:

- Android tiene decenas de versiones (API 14, 15, 16... 34).
- Cada API añade nuevas funciones.
- Si eliges un **Minimum SDK muy alto**, tu app funcionará en pocos móviles.
- Si eliges un **Minimum SDK muy bajo**, tienes más compatibilidad, pero **menos funciones modernas**.

Por eso la elección de API 24 es razonable:

— equilibrio perfecto entre compatibilidad y funciones recientes.

---

## 2.3 Build Configuration Language — Gradle y Kotlin DSL

Lo que ves como:

| Kotlin DSL (build.gradle.kts) — Recommended

Significa que tu proyecto usará **Kotlin DSL** en lugar de **Groovy** para los scripts de Gradle.

**Gradle** = herramienta que compila, empaqueta y firma tu aplicación.

- Kotlin DSL es más moderno.
- Android Studio lo recomienda por claridad y autocompletado.
- No afecta al código Java de tu app.

---

## 3. La anatomía real del proyecto Android (según el PDF + lo visto en clase)

El PDF lo estructura así:

1. AndroidManifest.xml
2. Layout XML
3. Activity Java
4. Clase R

5. strings.xml
6. Flujo de ejecución
7. Estructura de carpetas

Voy a ampliarlo y conectarlo con lo que el profesor muestra en tus capturas.

---

## 4. AndroidManifest.xml — El cerebro declarativo de la App

Código del PDF:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.ejemplo.miapp">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/Theme.MiApp">

        <activity
            android:name=".MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

### ¿Qué hace realmente cada línea?

#### 4.1 package="..."

Es el nombre único de tu app. Android lo usa para:

- instalar/desinstalar paquetes,
- generar la clase R,
- identificar la app internamente,
- separar tus recursos de los de otras apps.

#### 4.2 <application>

Agrupar todo lo referente a tu app:

- nombre visible,
- icono,
- tema visual,
- configuración de backup,
- Activities, Services, BroadcastReceivers...

#### 4.3 <activity>

Declara **cada pantalla** que tu app puede mostrar.

## 4.4 <intent-filter> con MAIN + LAUNCHER

Indica la pantalla **que aparece al abrir la app**.

Es el equivalente a “método main” del mundo Android.

## 5. Layout XML — La interfaz de usuario visual (activity\_main.xml)

Código del PDF:

```
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    android:padding="16dp"
    android:gravity="center">

    <TextView
        android:id="@+id/tvTitulo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Introduce tu nombre:"
        android:textSize="18sp"
        android:layout_marginBottom="16dp"/>

    <EditText
        android:id="@+id/etNombre"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:hint="Escribe aquí"
        android:inputType="textPersonName"
        android:layout_marginBottom="16dp"/>

    <Button
        android:id="@+id/btnSaludar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Saludar" />

</LinearLayout>
```

### 5.1 ¿Por qué XML?

Android utiliza XML porque permite **declarar** la interfaz de usuario de manera estructurada, legible y separada de la lógica.

No es solo una decisión estética: forma parte de una filosofía arquitectónica clara.

XML aporta:

- **Independencia entre UI y código:** puedes modificar el diseño sin tocar Java.

- **Compatibilidad con cientos de tamaños de pantalla:** Android escala automáticamente valores como `dp` y `sp`.
- **Inspección visual:** Android Studio puede generar previews sin compilar.
- **Herramientas automáticas:** temas, colores, traducciones, accesibilidad, etc.

Cuando el programador define un botón en XML, no está creando un botón real, sino **la descripción** de uno.

El elemento se materializa más tarde cuando Android lo convierte en objetos reales (ver apartado del `Inflater` más abajo).

Esa separación protege al programador de “ensuciar” el código Java con lógica de diseño, y viceversa. Es el mismo principio que se sigue en frameworks modernos (MVVM, MVC y derivados).

---

## 5.2 Los Layouts

Un layout es un **contenedor**. Controla cómo se distribuyen los elementos dentro de la pantalla. Cada tipo resuelve un problema diferente.

En este proyecto se usa **LinearLayout**, que organiza los elementos en una sola dirección (vertical en tu caso). Es el contenedor ideal para interfaces simples, como formularios o pantallas iniciales.

Aunque parezca trivial, la elección del layout influye directamente en:

- el rendimiento,
- la claridad de la interfaz,
- la escalabilidad futura.

Por eso Android recomienda comenzar por `LinearLayout` y luego avanzar a `ConstraintLayout` para diseños más complejos.

Los atributos del layout son esenciales:

- `orientation="vertical"` establece la dirección.
- `padding="16dp"` proporciona espacio interno consistente entre dispositivos.
- `gravity="center"` alinea el contenido en pantalla.

Estos valores son fundamentales porque Android trabaja sobre miles de resoluciones. Si usaras píxeles reales (px), tu UI se rompería en la mayoría de pantallas.

---

## 5.3 Atributos clave

En Android, cada vista debe tener **`layout_width`** y **`layout_height`**.

Los dos valores más habituales tienen un sentido profundo:

- **`match_parent`** → la vista ocupa todo lo que pueda en su contenedor
- **`wrap_content`** → la vista ocupa solo lo necesario para su contenido

Estos atributos definen el comportamiento espacial de cada elemento.

Combinarlos en función del contenedor permite construir interfaces robustas sin cálculos manuales.

Otro atributo fundamental es:

```
android:id="@+id/etNombre"
```

Este ID es la única forma que tiene Java de localizar ese elemento.

Sin ese ID, `findViewById()` no podría identificar qué vista debe devolver.

Todos los IDs se registran en la clase `R` automáticamente.

---

## 6. MainActivity.java — El controlador de la pantalla

### 6.1 Explicación en profundidad de lo que ocurre dentro de `onCreate()`

Este método es el **punto de entrada real** de la Activity.

No es un método cualquiera: es llamado directamente por Android como parte de su ciclo de vida.

Dentro de `onCreate()` suceden tres fases clave:

#### 1. Inicialización del contexto de la Activity

Android crea la instancia de tu Activity, pero aún no existe ninguna vista.

En este momento solo tienes acceso al estado (Bundle) y a la lógica interna.

#### 2. Inflado del layout con `setContentView()`

Este es el momento en que Android:

1. Lee el archivo XML
2. Interpreta cada etiqueta
3. Construye objetos concretos (`TextView`, `Button`, `EditText`, etc.)
4. Los coloca en memoria en forma de árbol de vistas
5. Los dibuja en pantalla

El programador solo ve una línea:

```
setContentView(R.layout.activity_main);
```

pero internamente Android ejecuta un proceso complejo para convertir XML → Objetos Java.

#### 3. Enlace entre la vista y la lógica

Aquí aparecen las llamadas:

```
etNombre = findViewById(R.id.etNombre);  
btnSaludar = findViewById(R.id.btnSaludar);
```

En ese momento, Java ya tiene acceso a las vistas generadas por el inflater.

Sin `setContentView()`, estas llamadas devolverían null.

#### 4. Registro del Listener (polimorfismo por interfaz)

Cuando haces:

```
btnSaludar.setOnClickListener(...)
```

no estás “programando un botón que reacciona”.

Estás registrando en Android un **objeto que implementa una interfaz** (`OnClickListener`).

Cuando el usuario toque el botón, Android ejecutará ese objeto.

Todo este proceso está basado en el modelo de eventos clásico de la POO.

---

## 7. Clase R — El puente entre XML y Java

La clase R no es un archivo escrito por el programador.

Es un archivo generado automáticamente durante la compilación.

Su función es actuar como **índice global** de todos los recursos:

- layouts
- strings
- IDs
- drawables
- colores
- estilos

Cuando aparece:

```
R.id.btnSaludar
```

Java no está “buscando un botón por nombre”.

Lo que recibe es un entero único que Android usa para localizar la vista correspondiente en la jerarquía de objetos creada por el inflater.

Este mecanismo permite que:

- Java, XML y recursos se mantengan sincronizados
- el código se rompa inmediatamente si borras un elemento
- no existan búsquedas lentas por nombre
- las apps sean más rápidas y fiables

Modificar la clase R manualmente no tiene sentido: se regeneraría en cuanto pulses “Run”.

---

## 8. strings.xml — Buenas prácticas desde el principio

Aunque esta parte parece menor, es esencial para un desarrollo profesional.

strings.xml permite:

- separar el contenido textual del código
- administrar múltiples idiomas
- facilitar accesibilidad (TalkBack lee textos definidos correctamente)
- evitar duplicación de cadenas
- permitir que diseñadores o traductores trabajen sin tocar Java

Si mañana decides añadir catalán, inglés o francés, solo necesitas duplicar la carpeta:

```
values-ca/  
values-en/  
values-fr/
```

Y Android seleccionará automáticamente el idioma del sistema.

---

## 9. Flujo de ejecución real según Android

Android no ejecuta tu app como si fuera un programa Java normal.

Sigue un proceso rígido diseñado para proteger el sistema y optimizar recursos.

Cuando el usuario toca el icono:

1. Android consulta **AndroidManifest.xml**
2. Busca la Activity marcada como LAUNCHER
3. Crea la instancia de esa Activity
4. Llama a los métodos del ciclo de vida en orden

Cada método ( `onCreate()` , `onStart()` , `onResume()` ) corresponde a un estado interno del sistema: inicialización, preparación y visualización.

El usuario solo ve una pantalla, pero Android está gestionando en segundo plano:

- memoria
- entrada del teclado
- sensores
- animaciones
- restauración del estado
- posible destrucción de la Activity si el sistema necesita RAM

Este flujo es el que explica por qué Android es tan sólido incluso en dispositivos de gama baja.

---

## 10. Estructura de Carpetas

```
MiApp/
|
├── app/
|   ├── src/
|   │   ├── main/
|   │   │   ├── java/
|   │   │   │   ├── com/ejemplo/miapp/
|   │   │   │   └── MainActivity.java
|   │   │   ├── res/
|   │   │   │   ├── layout/
|   │   │   │   │   └── activity_main.xml
|   │   │   │   ├── values/
|   │   │   │   │   └── strings.xml
|   │   │   │   ├── mipmap/
|   │   │   │   │   └── ic_launcher.png
|   │   │   └── AndroidManifest.xml
|   └── build.gradle (configuración de la app)
└── build.gradle (configuración del proyecto)
```

### Explicación profunda de la estructura

La estructura de carpetas de un proyecto Android está diseñada para separar claramente **código**, **recursos**, **configuración** y **metadatos**. Esta separación no es arbitraria: sigue principios de arquitectura software (MVC/MVVM) y permite que Android gestione eficientemente la app en tiempo de ejecución.

## 10.1 Carpeta `/app/src/main/java/`

Esta carpeta contiene **todo el código Java o Kotlin** de tu aplicación.

- En su interior encontrarás un paquete con el nombre que definiste en el wizard (ej.: `com.ejemplo.miapp`).
- Cada Activity, Fragment, clase de utilidad, repositorio, ViewModel, etc., vive aquí.
- La organización en paquetes ayuda a mantener el proyecto escalable y coherente.

Ejemplo:

```
com.ejemplo.miapp/  
└─ MainActivity.java
```

Este archivo define **el comportamiento** de la pantalla principal.

No contiene nada visual: solo lógica.

---

## 10.2 Carpeta `/app/src/main/res/`

Es el corazón visual de la app. Aquí se almacena **todo lo que no es código Java**, es decir, todos los **recursos**:

- layouts
- textos
- colores
- imágenes
- estilos
- iconos

Android accede a cada uno mediante la clase `R`.

---

## 10.3 `/res/layout/`

Contiene los **archivos XML** que describen cómo se verá cada pantalla.

Ejemplo:

```
activity_main.xml
```

Este archivo describe la interfaz: botones, textos, posiciones, tamaños, márgenes...

Android lo convierte en objetos reales mediante un proceso llamado **inflado del layout**.

---

## 10.4 `/res/values/`

Aquí se guardan valores reutilizables:

- Strings ( `strings.xml` )
- Colores ( `colors.xml` )

- Estilos ( `styles.xml` )
- Dimensiones ( `dimens.xml` )

Ejemplo:

```
<string name="titulo">Introduce tu nombre:</string>
```

Gracias a esta carpeta, tu app soporta fácilmente **múltiples idiomas** y mantiene el código limpio.

---

## 10.5 /res/mipmap/

Almacena los iconos de la app en diferentes resoluciones.

Android selecciona automáticamente el icono más adecuado según el tipo de pantalla, para que el launcher no se vea pixelado.

Ejemplo clásico:

```
ic_launcher.png
```

## 10.6 Archivo `AndroidManifest.xml`

Es la **pieza central** de configuración de tu app.

Define:

- qué Activities existen
- cuál es la pantalla inicial (LAUNCHER)
- qué permisos requiere tu app (cámara, internet, etc.)
- el nombre del paquete
- metadatos internos del sistema

En cierto modo, es el “DNI” de tu aplicación dentro de Android.

---

## 10.7 Archivos Gradle ( `build.gradle` )

Tienes dos niveles:

### a) `build.gradle` (módulo app)

- define dependencias
- SDK mínimo y objetivo
- versión de la app
- bibliotecas de terceros (Gson, Retrofit, Firebase)

### b) `build.gradle` (proyecto raíz)

- configuración global
- versiones comunes
- repositorios ( `maven` , `google` )

Gradle es el “motor de construcción” que se encarga de compilar, unir recursos, minificar el código y generar el APK final.

---

# 11. Conceptos Clave para Recordar

## 11.1 Activity — Una pantalla de tu app

Una Activity representa **una única pantalla completa en Android**.

Es comparable a una ventana en escritorio, pero mucho más controlada porque Android gestiona su ciclo de vida:

- creación
- visibilidad
- pausa
- reanudación
- destrucción

La Activity es la responsable de:

- cargar el layout
- gestionar eventos del usuario
- coordinar datos y vistas

---

## 11.2 Layout XML — Define CÓMO se ve la pantalla

XML describe la parte visual.

Cuando escribes un `<Button>` o un `<TextView>`, no estás creando un botón real: estás definiendo su estructura.

Android usa esta descripción para construir los objetos visuales en memoria y mostrarlos al usuario.

---

## 11.3 Código Java — Define QUÉ hace la app

Mientras XML describe “qué se ve”, Java describe “qué pasa”.

Ejemplos:

- qué ocurre al pulsar un botón
- cómo validar datos del formulario
- qué lógica ejecuta la app
- cómo se comunica con el sistema Android

La Activity actúa como “controlador” y es el intermediario entre vistas (XML) y datos.

---

## 11.4 R.java — El puente entre código y recursos XML

`R` es una clase generada automáticamente que contiene **IDs únicos** para todos los recursos del proyecto.

Ejemplos:

```
R.layout.activity_main
R.id.btnSaludar
R.string.app_name
```

Cuando llamas a `findViewById(R.id.tvTitulo)` , Android busca exactamente ese objeto visual correspondiente al ID generado.

Sin `R` , Android no podría enlazar tu layout con la Activity.

---

## 11.5 Toast — Mensaje emergente temporal

Un Toast es un pequeño mensaje flotante que informa al usuario sin interrumpir su interacción.

Ejemplo:

```
Toast.makeText(this, "Hola!", Toast.LENGTH_SHORT).show();
```

Se usa para:

- notificaciones cortas
- validaciones rápidas
- mensajes no críticos

No detiene la app, no bloquea la pantalla y desaparece solo.

---